

# Lexicographic identifying codes

Maximilien Gadouleau  
 School of Engineering and Computing Sciences  
 Durham University  
 m.r.gadouleau@durham.ac.uk

April 10, 2013

## Abstract

An identifying code in a graph is a set of vertices which intersects all the symmetric differences between pairs of neighbourhoods of vertices. Not all graphs have identifying codes; those that do are referred to as twin-free. In this paper, we design an algorithm that finds an identifying code in a twin-free graph on  $n$  vertices in  $O(n^3)$  binary operations, and returns a failure if the graph is not twin-free. We also determine an alternative for sparse graphs with a running time of  $O(n^2 d \log n)$  binary operations, where  $d$  is the maximum degree. We also prove that these algorithms can return any identifying code with minimum cardinality, provided the vertices are correctly sorted.

## 1 Introduction

Identifying codes were introduced in [1] for fault diagnosis in multiprocessor systems, and have since then found applications in location and detection problems. In general, an identifying code in a graph  $G$  can be defined as follows. First, we denote the (closed) neighborhood of any vertex  $v$  as  $N(v) = \{v\} \cup \{w : vw \in E(G)\}$ . An *identifying code* is a subset of vertices which satisfies the following property: for any two vertices  $v$  and  $w$ , we have  $N(v) \cap C \neq N(w) \cap C \neq \emptyset$ . Equivalently, it is any subset of vertices  $C$  such that for all  $v_1, v_2 \in V(G)$ ,  $(N(v_1) \Delta N(v_2)) \cap C \neq \emptyset$ , where  $\Delta$  is the symmetric difference between two sets. A graph admits an identifying code if and only if it is *twin-free* [2], where twins are two vertices with the same neighborhood. We remark that the definitions above are commonly used for a so-called 1-identifying code, where an  $r$ -identifying code is defined in terms of balls of radius  $r$  around a vertex. Since any  $r$ -identifying code can be seen as a 1-identifying code for a related graph, we do not lose any generality in considering 1-identifying codes only. For a thorough survey of identifying codes, the reader is invited to [3], and an exhaustive literature bibliography on identifying codes and related topics is maintained in [4].

Since any superset of an identifying code is itself an identifying code, it is natural to search for the minimum cardinality  $i(G)$  of an identifying code of a given graph  $G$ . Let us refer to an identifying code as *minimal* if it has no proper subset which itself is an identifying code and as *minimum* if it has the smallest cardinality amongst all codes. The problem of finding the minimum cardinality of an identifying code was shown to be NP-hard in [3]. Viewing this problem as an instance of the subset cover problem [5], a greedy heuristic was also designed and analyzed in [3]. Its running time is on the order of  $O(n^4)$  binary operations and has the following performance guarantees. It always finds an identifying code whose cardinality is less than  $c_1 i(G) \ln n$  for some nonnegative constant  $c_1$ ; however, there are graphs for which the algorithm always returns a code with cardinality greater than  $c_2 i(G) \ln n$  for another nonnegative constant  $c_2$ .

*Lexicographic codes* were introduced in [6] and independently rediscovered in [7] to design large constant-weight codes, which are sets of binary vectors of equal Hamming weight with a prescribed minimum Hamming distance (see [8] for a detailed review of constant-weight codes and lexicographic codes). The principle is to first sort all the vectors with the same Hamming weight, and then construct the code as we run through them. Adding a codeword is done according to a simple criterion: it must be at distance at least  $d$  from

the code constructed so far. The performance of the algorithm depends on the order in which the vectors have been sorted; moreover, some modifications can be added, such as starting with a predetermined set of vectors. Many record-holding constant-weight codes have been designed using lexicographic codes. However, this idea is not limited to constant-weight codes, and their application to nonrestricted binary codes has led to many interesting results [9]. They have also been recently applied to the construction of codes on subspaces in [10], also yielding record-holding codes.

In this paper, we investigate adapting the idea of lexicographic codes to identifying codes. The main contribution is an algorithm running in  $O(n^3)$  binary operations which returns an identifying code for a twin-free graph, and returns a failure if the graph is not twin-free. This algorithm is then adapted to sparse graphs to run in  $O(n^2 d \log n)$  binary operations. Both algorithms have the same guarantees in terms of cardinality of the output. Although we are unable to give an upper bound which does not depend on the ordering of the vertices, we show that provided the vertices are properly sorted, the algorithm returns a minimum identifying code. This is fundamentally different to the greedy approach in  $O(n^4)$ .

## 2 Algorithm for general graphs

### 2.1 Description and correctness

Let  $G$  be a graph on  $n$  vertices with adjacency matrix  $\mathbf{A}$ , and let  $\mathbf{B} = \mathbf{I}_n + \mathbf{A}$ . We denote the vertices as  $v_1, v_2, \dots, v_n$ , thus  $b_{i,j} = 1$  if and only if  $v_i \in N(v_j)$ ; yet we shall abuse notation and identify a vertex with its index. For instance, we refer to the vertex with minimum index in the neighborhood of  $v_i$  as  $\min 1(i)$ . Also, the output of our algorithm is actually the set of indices of the vertices in the code.

Before giving the pseudocode of Algorithm 1, we describe it schematically below. Its input is the matrix  $\mathbf{B}$  of the graph. It then runs along all vertices  $v_j$ , adding a new codeword to the code  $C$  if  $N(v_j) \cap C = \emptyset$  or  $N(v_j) \cap C = N(v_k) \cap C$  for some  $k < j$ . While searching for a new codeword to add, the algorithm may return a failure if the graph is not twin-free, which we identify as  $n+1 \in C$ . After the  $j$ -th step, the code  $C$  then ‘identifies’ the first  $j$  vertices, i.e. they are all covered in a distinct fashion. We keep track of the intersections  $N(v_i) \cap C$  in a matrix  $\mathbf{X}$ . After going through all vertices, the algorithm then returns an identifying code  $C$  or a failure (if  $n+1 \in C$ ) if the graph is not twin-free.

The subroutine  $\min 2(j, k)$  returns the first vertex which identifies  $v_j$  if it exists and a failure otherwise, i.e. it determines the first vertex in lexicographic order in  $N(v_j) \Delta N(v_k)$ . If  $N(v_j) = N(v_k)$ , then it returns  $n+1$ . It is given in Algorithm 2.

We now justify this claim in Lemma 1 below.

**Lemma 1** *The subroutine  $\min 2(j, k)$  returns the minimum element in  $N(v_j) \Delta N(v_k)$  if this symmetric difference is non-empty, and a failure ( $l = n+1$ ) otherwise.*

**Proof** First, if  $N(v_j) = N(v_k)$ , then  $\mathbf{B}(j, l) = \mathbf{B}(k, l)$  for all  $1 \leq l \leq n$ . Therefore, the **while** loop will only stop once  $l = n+1$ , and hence the subroutine returns a failure. Second, if  $N(v_j) \neq N(v_k)$ , then the minimum element in  $N(v_j) \Delta N(v_k)$  is the smallest  $l$  such that  $\mathbf{B}(j, l) \neq \mathbf{B}(k, l)$ . It is clear that the subroutine returns this value.  $\square$

**Proposition 1** *Algorithm 1 returns an identifying code if the input graph is twin-free, and a failure ( $n+1 \in C$ ) otherwise.*

**Proof** First of all, we prove that the algorithm returns a failure if and only if the graph is not twin-free. In the latter case, let  $k$  be the smallest integer such that the set  $\{i \neq k : N(v_k) = N(v_i)\}$  is not empty, and let  $j$  be the minimum element of this set (hence  $k < j$ ,  $N(v_k) = N(v_j)$ ). It is easily shown that after the  $k$ -th step,  $v_k$  is covered. On the  $j$ -th step, Algorithm 1 first checks if  $v_j$  is covered. Since  $v_k$  is covered and  $N(v_k) = N(v_j)$ , then  $v_j$  is also covered. Algorithm 1 then finds that  $k$  is the smallest integer satisfying  $\mathbf{X}(k) = \mathbf{X}(j)$ , and hence calls the subroutine  $\min 2(j, k)$ . By Lemma 1 this returns a failure, and hence the whole algorithm returns a failure. Conversely, the only case where the subroutine (and hence the algorithm) returns a failure is when there exist  $k < j$  such that  $N(v_j) = N(v_k)$ , i.e. the graph is not twin-free.

---

**Algorithm 1** Main algorithm for general graphs

---

```
 $C \leftarrow \emptyset, X \leftarrow \mathbf{0}_n, j \leftarrow 1$ 
while  $j \leq n$  and  $n + 1 \notin C$  do
   $l \leftarrow 0$ 
  if  $\mathbf{X}(j) = \mathbf{0}$  then  $\{v_j \text{ is not covered}\}$ 
     $l \leftarrow \text{min1}(j)$ 
  else
     $k \leftarrow 1$ 
    while  $\mathbf{X}(j) \neq \mathbf{X}(k)$  and  $k < j$  do  $\{v_j \text{ is covered, so we search if it is identified}\}$ 
       $k \leftarrow k + 1$ 
    end while
    if  $k < j$  then  $\{v_j \text{ is not identified}\}$ 
       $l \leftarrow \text{min2}(j, k)$ 
    end if
  end if
  if  $1 \leq l \leq n$  then  $\{\text{A new codeword has been found}\}$ 
     $C \leftarrow C \cup \{l\}$ 
     $\mathbf{X}^T(l) \leftarrow \mathbf{B}^T(l)$ 
  end if
   $j \leftarrow j + 1$ 
end while
return  $C$ 
```

---

---

**Algorithm 2**  $\text{min2}(j, k)$  subroutine

---

```
 $l \leftarrow 1$ 
while  $l \leq n$  and  $\mathbf{B}(j, l) = \mathbf{B}(k, l)$  do
   $l \leftarrow l + 1$ 
end while
return  $l$ 
```

---

We now assume that the graph is twin-free, and hence we have  $l \leq n$  at any step. We need to show that the output  $C$  of Algorithm 1 is an identifying code. Let us denote the matrix  $\mathbf{X}$  and the code  $C$  obtained after  $j$  steps as  $\mathbf{X}^j$  as  $C^j$ , respectively. Note that for all  $a$ ,  $\mathbf{X}^j(a)$  reflects how the vertex  $v_a$  is covered by  $C^j$ :  $N(v_a) \cap C^j = \text{supp}(\mathbf{X}^j(a)) = \{b : \mathbf{X}^j(a, b) = 1\}$ . The following claim is the cornerstone of the proof.

**Claim:** After step  $j$ , all  $\mathbf{X}^j(i)$ 's are nonzero and distinct for  $1 \leq i \leq j$ .

The proof goes by induction on  $j$ , and is trivial for  $j = 1$ . Suppose it is true for  $j - 1$ , then

$$\text{supp}(\mathbf{X}^{j-1}(a)) = N(v_a) \cap C^{j-1} \subseteq N(v_a) \cap C^j = \text{supp}(\mathbf{X}^j(a)). \quad (1)$$

It is hence easy to show that if  $\mathbf{X}^{j-1}(a) \neq \mathbf{0}$ , then  $\mathbf{X}^j(a) \neq \mathbf{0}$  and if  $\mathbf{X}^{j-1}(a) \neq \mathbf{X}^{j-1}(b)$ , then  $\mathbf{X}^j(a) \neq \mathbf{X}^j(b)$  for all  $a$  and  $b$ . It immediately follows that the vectors  $\mathbf{X}^j(i)$ 's are all nonzero and distinct for  $1 \leq i \leq j - 1$ , and we only have to consider  $\mathbf{X}^j(j)$ . Three cases occur when the algorithm reaches step  $j$ .

- Case I:  $\mathbf{X}^{j-1}(j)$  is nonzero and distinct to any  $\mathbf{X}^{j-1}(i)$  for  $1 \leq i \leq j - 1$ . Then as shown above,  $\mathbf{X}^j(j)$  is nonzero and distinct to all  $\mathbf{X}^j(i)$ 's.
- Case II:  $\mathbf{X}^{j-1}(j)$  is nonzero and equal to  $\mathbf{X}^{j-1}(k)$  for some  $k < j$ . First, we remark that  $k$  is unique, as  $\mathbf{X}^{j-1}(k) \neq \mathbf{X}^{j-1}(i)$  for all other  $i$ . The  $\text{min2}(k, j)$  subroutine then returns an element  $v_l \in N(v_j) \Delta N(v_k)$ , and hence  $\mathbf{X}^j(j, l) \neq \mathbf{X}^j(k, l)$ .
- Case III:  $\mathbf{X}^{j-1}(j) = \mathbf{0}$ . Then by hypothesis  $\mathbf{X}^{j-1}(j) \neq \mathbf{X}^{j-1}(i)$  for all  $1 \leq i \leq j - 1$ , and hence  $\mathbf{X}^j(j) \neq \mathbf{X}^j(i)$ . Also,  $\mathbf{X}^j(j)$  is the unit vector  $\mathbf{e}_{\text{min1}(j)}$ , which is nonzero.

Therefore, for the code  $C^n = C$  obtained after  $n$  steps,  $N(v_a) \cap C$  are all nonzero and distinct for all  $1 \leq a \leq n$ . It is hence an identifying code.  $\square$

## 2.2 Performance

We now investigate the performance of Algorithm 1. We are first interested in the cardinality of its output. Clearly, this depends on the order in which the vertices are sorted. We show below that provided the order is suitable, the algorithm can find any minimal identifying code, and hence can return a minimum one.

**Proposition 2** *Suppose that the graph is twin-free and that  $M = \{v_1, v_2, \dots, v_m\}$  forms an identifying code. Then Algorithm 1 returns an identifying code that is a subset of  $M$ .*

**Proof** We know by Proposition 1 that the algorithm returns an identifying code; we only have to prove that all codewords are in  $M$ . At step  $j$ , three cases need to be distinguished.

- Case I:  $v_j$  is covered and identified, then no codeword is added.
- Case II:  $v_j$  is covered but not identified, i.e.  $(N(v_j) \Delta N(v_k)) \cap C^{j-1} = \emptyset$  for some  $k < j$ . The subroutine returns the smallest element  $v_l$  in  $N(v_j) \Delta N(v_k)$ . Since  $M$  is an identifying code, the set  $(N(v_j) \Delta N(v_k)) \cap M$  is not empty, hence  $v_l \in M$ .
- Case III:  $v_j$  is not covered. The algorithm then selects the next codeword to be  $\text{min1}(j)$ , which is necessarily in  $M$  as  $N(v_j) \cap M \neq \emptyset$ .

Therefore, the algorithm only adds codewords of  $M$ , and hence returns a subcode of  $M$ .  $\square$  We remark that Algorithm 1 does not necessarily return a minimal code, as seen in Figure 1. Algorithm 1 would return the code  $\{1, 2, 3, 4, 5, 6\}$  while  $\{2, 3, 4, 5, 6\}$  is a minimal identifying code.

On the other hand, if  $M$  is minimal, then it has no proper subset that itself is an identifying code; Algorithm 1 thus returns it. We obtain the following corollary.

**Corollary 1** *Provided that the vertices are sorted such that  $v_1, v_2, \dots, v_m$  form a minimal identifying code for some  $1 \leq m \leq n$ , Algorithm 1 will return this identifying code.*

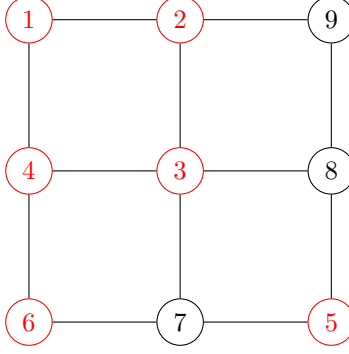


Figure 1: A graph and a sorting of vertices such that the lexicographic code is not minimal

Proposition 2 also implies that the probability that the output has cardinality no more than  $K$  is at least the probability that the first  $K$  vertices form an identifying code. Hence our algorithm returns a minimum identifying code with probability at least  $\frac{1}{\binom{n}{i(G)}}$ .

**Proposition 3** *The running time of Algorithm 1 is  $O(n^3)$  binary operations.*

**Proof** Clearly, we have to run the iteration for  $j$  exactly  $n$  times. For each iteration, the step demanding the highest number of operations is the search for  $k$ . We consider at most  $j - 1$  values of  $k$ , comparing at most  $n$  bits to verify whether  $\mathbf{X}(j) \neq \mathbf{X}(k)$ . Therefore, the running time is  $O(n^3)$ .  $\square$

### 3 Algorithm for sparse graphs

For sparse graphs, it is more efficient not to work with the whole adjacency matrix, but with the neighborhood array  $A \in \mathcal{P}(E)^n$ , defined as  $A(v_i) = N(v_i)$ , where the neighborhood is sorted in increasing lexicographic order. Then, instead of adding the column of the adjacency matrix corresponding to a new codeword, we only update the code array  $X(v)$  for all vertices adjacent to the new codeword. The algorithm for sparse graphs is given in Algorithm 3; its input is the neighborhood array, and it returns an identifying code  $C$  or a failure ( $n + 1 \in C$ ) if the graph is not twin-free.

Similar to the general case, the  $\text{min3}(j, k)$  subroutine produces the first vertex  $v_l$  which identifies  $v_j$  if it exists and a failure otherwise, i.e. it determines the first vertex in lexicographic order which covers either  $j$  or  $k$ , but not both. It is given in Algorithm 4.

The same results on correctness and the possibility of returning a minimum code also hold for Algorithm 3; they are summarized below.

**Proposition 4** *If the graph is not twin-free, then Algorithm 3 returns a failure. Otherwise, the algorithm returns an identifying code contained in  $\{v_1, v_2, \dots, v_m\}$ , where  $m$  is the minimum integer such that this forms an identifying code.*

*The running time of Algorithm 3 is  $O(n^2 d \log n)$  binary operations.*

**Proof** The proof of correctness of Algorithm 3 is similar to that of Algorithm 1, and is hence omitted. We hence determine the running time of the algorithm.  $\square$

## References

- [1] M. G. Karpovsky, K. Chakrabarty, and L. B. Levitin, “A new class of codes for identification of vertices in graphs,” *IEEE Trans. Info. Theory*, vol. 44, no. 2, pp. 599–611, March 1998.

---

**Algorithm 3** Main algorithm for sparse graphs

---

```
 $C \leftarrow \emptyset, X \leftarrow \emptyset^n, j \leftarrow 1, f \leftarrow 0$ 
while  $j \leq n$  and  $n + 1 \notin C$  do
   $l \leftarrow 0$ 
  if  $X(j) = \emptyset$  then  $\{v_j \text{ not covered}\}$ 
     $l \leftarrow A(j, 1)$ 
  else
     $m \leftarrow X(j, 1), k \leftarrow 1$ 
    while  $X(j) \neq X(k)$  and  $k < j$  do
       $k \leftarrow k + 1$ 
    end while
    if  $k < j$  then  $\{v_j \text{ not identified}\}$ 
       $l \leftarrow \text{min3}(j, k)$ 
    end if
  end if
  if  $1 \leq l \leq n$  then
     $C \leftarrow C \cup \{l\}$ 
    for  $i$  from 1 to  $d_l$  do
       $X(A(l, i)) \leftarrow X(A(l, i)) \cup \{l\}$ 
    end for
  end if
   $j \leftarrow j + 1$ 
end while
return  $C$ 
```

---

---

**Algorithm 4**  $\text{min3}(j, k)$  subroutine

---

```
 $l \leftarrow n + 1$ 
while  $a \leq \min\{d_j, d_k\}$  do
  if  $A(j, a) \neq A(k, a)$  then
     $l \leftarrow \min\{A(j, a), A(k, a)\}$ 
  end if
   $a \leftarrow a + 1$ 
end while
if  $l = n + 1$  then
  if  $d_j < d_k$  then
     $l \leftarrow A(k, d_j + 1)$ 
  else if  $d_k < d_j$  then
     $l \leftarrow A(j, d_k + 1)$ 
  end if
end if
return  $l$ 
```

---

- [2] I. Charon, I. Honkala, O. Hudry, and A. Lobstein, “Structural properties of twin-free graphs,” *The Electronic Journal of Combinatorics*, vol. 14, no. 1, p. R16, January 2007.
- [3] M. Laifenfeld and A. Trachtenberg, “Identifying codes and covering problems,” *IEEE Trans. Info. Theory*, vol. 54, no. 9, pp. 3929–3950, September 2008.
- [4] A. Lobstein. Watching systems, identifying, locating-dominating and discriminating codes in graphs. [Online]. Available: <http://www.infres.enst.fr/~lobstein/bibLOCDOMetID.html>
- [5] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*. MIT Press, 2001.
- [6] V. Levenshtein, “A class of systematic codes,” *Soviet Math. Dokl. 1*, pp. 368–371, 1960.
- [7] J. H. Conway and N. J. A. Sloane, “Lexicographic codes: error-correcting codes from game theory,” *IEEE Trans. Info. Theory*, vol. 32, pp. 337–348, May 1986.
- [8] A. E. Brouwer, J. B. Shearer, N. J. A. Sloane, and W. D. Smith, “A new table of constant weight codes,” *IEEE Trans. Info. Theory*, vol. 36, no. 6, pp. 1334–1380, November 1990.
- [9] A. Trachtenberg, “Error-correcting codes on graphs: Lexicodes, trellises, and factor graphs,” Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2000.
- [10] N. Silberstein and T. Etzion, “Large constant-dimension codes and lexicodes,” in *Proc. Algebraic Combinatorics and Applications*, Thurnau, Germany, April 2010.